TEC | Tecnológico de Costa Rica

Gramática de un lenguaje loco

Lenguaje para discapacitados

IC-5701 **Compiladores E Intérpretes**

Sánchez Sánchez Miguel David - 2019061555

Sánchez Vargas Kaled - 2019160584

Lumbi Murillo Kimberly Verónica - 2020146765

Vargas Fletes Juan José - 2020035292

Picado Arias Andrey Fabián- 2020135773

TEC Costa Rica, Cartago

Prof. Aurelio Sanabria, grupo 2

II Semestre 2022 – 26 de agosto

Tabla de contenidos

Motivación

El lenguaje de programación “Búho” está diseñado para todas aquellas personas no videntes amantes de escribir código.

La inspiración del nombre nace de la capacidad auditiva de este animal, que incluso lo posiciona como "El rey de las aves en cuanto a sentido del oído" (Instituto ORL-IOM, 2016). Tienen este sentido tan agudo, que se valen del mismo para rastrear la posición exacta de una presa, por pequeña que sea, y atacarla; sentido que, dicho sea de paso, suple la carencia de visión que estas aves tienen, ya que, según el Dr. Giménez, en su artículo “¿Sabías que… los búhos son hipermétropes?” (19 julio, 2017) lo son, (De acuerdo con la American Academy of Ophthalmology, en resumen, la hipermetropía es una condición que implica dificultad para ver de cerca), lo cual es una noble y respetuosa comparación con las personas con discapacidad visual.

Es importante destacar que el hecho de que los búhos no gozan de una gran capacidad visual, es un dato poco popular, y que por el contrario, es un animal admirado y considerado majestuoso por las personas. Esto último, es precisamente la razón de que nuestro lenguaje de programación lleve el nombre de este animal, ya que el motivo del mismo, es facilitar a que los programadores no videntes enaltezcan a través del código su talento y capacidad para programar de una manera cómoda y accesible, y a su vez, que su identidad deje de ser reducida a su discapacidad.

El lenguaje de programación Búho, busca facilitar la programación a las personas no videntes a través de una sintaxis sencilla, que al ser leída por los diferentes lectores de pantalla comúnmente utilizados por las personas no videntes, sea narrado de una forma parecida a la lectura en lenguaje natural. Es posible encontrar palabras reservadas que reemplazan signos de puntuación para una lectura más fluida de la misma. Por ejemplo "tiene" en lugar de "=", o "menor_igual" en lugar de "<="; considerando a su vez que es más sencillo para una persona no vidente ubicar las letras en el teclado, que los signos de puntuación, cuya localización puede variar dependiendo del idioma en el que la computadora está configurada.

En resumen, Búho es un lenguaje de programación en español, fácil de leer tanto para una persona como para un lector de pantallas. Fue concebido y diseñado pensando en las personas con dificultades visuales que hacen uso de un lector de pantallas para interactuar con el dispositivo que programan, y su objetivo principal es buscar ser accesible al procurar que los lectores de pantalla lean el código de una forma que suene lo más parecido posible al lenguaje natural, evitando pronunciar los caracteres especiales tan populares entre los lenguajes de programación comunes.

Análisis del lenguaje

Al ser un lenguaje general desarrollado para una población específica tiene puntos fuertes y débiles que se presentan cuando hablamos del lenguaje para su uso general y para cuando se usa en su propósito original. Entre los puntos fuertes de nuestro lenguaje se encuentran:

- Se centra en la abolición del uso de signos, nuestro lenguaje se hace fácil de leer por los “lectores de pantalla” que utilizan generalmente nuestra población objetivo.
- Es más fácil de entender para alguien que está empezando a programar por su sencillez y ser de alto nivel.
- La estructura del lenguaje permite una fácil comprensión por parte de personas ajenas a la programación, debido a que intenta seguir la estructuras de las oraciones de español.

Entre los puntos débiles que presenta el lenguaje Búho:

- Al ser un lenguaje que utiliza únicamente palabras claves, el lenguaje se vuelve tedioso de escribir debido a que las líneas de código se vuelven rápidamente largas.
- Al evitar el uso de signos, como paréntesis, las operaciones aritméticas se vuelven aún más largas y difíciles de diseñar. Para ejemplificar este caso se mostrará la comparación de un código equivalente en python y otro en Búho:

Python	Búho

`num = 0`
`x = 10`
`y = 50`
`num = 10*(x+(y/5))`
	`flotante num tiene 0`
	`numerico x tiene 10`
	`numerico y tiene 50`
	`num tiene y entre 5`
	`num tiene x mas num`
	`num tiene 10 por num`

- Al utilizar el signo de “_” para separar las palabras claves debido a una limitación en los lectores de pantalla para leer mayúsculas, nuestro lenguaje se vuelve un poco más complicado de escribir para las personas no videntes.
- Nuestro lenguaje utiliza muchas palabras reservadas para las palabras reservadas, lo que dificulta la memorización de todas estas y que el programador las use como variables (como puede ser numérico y tiene 0)

Puntos interesantes o chistosos:

- Aunque nuestro lenguaje intentó eliminar completamente los símbolos, se utilizan 3 símbolos para mejorar la comprensión del lenguaje. Estos símbolos son la “,” para los flotantes, la “_” para separar palabras claves y el “ ”” ” para encerrar los strings o textos del programa.
- Se pensó originalmente en incluir los números como sus palabras (1 = “uno”), sin embargo, la idea se descartó ya que complicaría a gran manera la identificación de números en cualquier parte del código.
- Aunque nuestro lenguaje está diseñado para ser leído en un lector de pantalla, al estar diseñado en español, se oye extraña la narración debido a la falta de tildes en las palabras claves.
- Se pensó utilizar una palabra reservada para delimitar el final de las instrucciones, el cual era “.”. Sin embargo, la idea se descartó porque el código se veía poco agradable a la vista y el lector de pantalla no los leía.
- Las funciones de ambiente estándar “dormir”, “aleatorio” y “valor_absoluto” se agregaron pensando en el uso futuro del lenguaje para programas más flexibles o interesantes
- La extensión de los archivos del lenguaje se definió como “bh” que se asemeja a búho.

Gramática EBNF

```
//Inicio de programa, Programa se conforma por secciones en las que puede estar compuesto
//Todos los inicios de los bloques de código no tienen puntos. El resto de las líneas terminan en punto.
Programa::= (Comentario | Declaracion | Expresion | Operandos | Funcion | AccesoDatosComplejos)*

//Programa ignorará todo en Comentario hasta el primer salto de línea
Comentario ::= "//" (\w* | \s*)*$

//Declaracion define un tipo de dato y le da un valor
//Si Expresion se usa sin Declaracion antes, este toma un Identificador ya declarado anteriormente y le otorga un nuevo valor
u Operacion (o mejor dicho el resultado que retorne dicha Operacion)
Declaracion ::= ( DeclaracionComun | DeclaracionMapa | DeclaracionLista )
DeclaracionComun ::= Tipo Expresion "\n"
     Tipo ::= ("numerico" | "flotante" | "texto" | "bool" | "lista" | "mapa")
DeclaracionMapa ::= "mapa" Identificador "de" Tipo "a" Tipo "\n"
DeclaracionLista ::= "lista" Identificador "de" Tipo "\n"
```

```
//La Expresion es la asignación de valores sencillos a variables
Expresion::= Identificador "tiene" (Operacion | Valor) "\n"
     Identificador ::= [a-z]+
     Operacion ::= Valor (OperadoresArtimeticos | OperadoresLogicos) Valor
          OperadoresArtimeticos ::= ("mas" | "menos" | "por" | "entre" | "residuo" | "elevado" | "modulo" )
          OperadoresLogicos::= ("menor" | "mayor" | "menor_igual" | "mayor_igual" | "diferente" | "igual" | "y" | "o" |
          "no")

     Valor ::= (Identificador | Numero | Flotante | Texto | Booleano | Llamada | LlamarMapa | Aleatorio)
          Numero ::= (-)?\d+
          Flotante ::= (-)?([0-9]*[,])[0-9]+
          Texto ::= ^(\").+(\")$
          Booleano ::= ("verdadero" | "falso")
          Lista ::= "de" Tipo
          Mapa ::= "de" Tipo "a" Tipo
          Llamada ::= "llamar" Identificador ("recibe" Parametro+)?
          Aleatorio::= "numerico_aletorio" ("de" Numero "a" Numero)?
```

```
//Los Operandos son las funciones básicas del sistema, no confundir con los operadores
Operandos::= (Escribir | Recibir_entrada | Si | Mientras | Desde | Dormir | ValorAbsoluto) "\n"
     Escribir ::= "escribir" (Booleano | Numero | Texto | Identificador)*
     Recibir_entrada ::= "recibir_entrada" ("con_comentario" (Booleano | Numero| Texto) | "sin_comentario" ) Guardar_en
          Guardar_en ::= "guardar_en" Identificador

     Si ::= "si" Condicion "inicio_si" Instruccion+ "final_si" ("\n" Sino)?
          Sino ::= "sino" Instruccion+ "final_sino"
          //Múltiple condición siempre compara de izquierda a derecha, no utiliza paréntesis para prioridad
          Condicion ::= Comparacion (("y" | "o")| Comparacion)?
          Comparacion ::= Comparador OperadoresLogicos Comparador
          Comparador ::= (Identificador | Booleano | Numero | Flotante | Texto)

     Mientras ::= "mientras" Condición "inicia_mientras" Instruccion+ "final_mientras"

     //Desde siempre se detendrá hasta llegar al segundo Numero, requiere que se coloque la operación para llegar a este
     //segundo Numero dentro de la Instrucción o de otro modo el ciclo no terminará
     Desde ::= "desde" Numero "hasta" Numero "inicia_desde" Instruccion+ "final_desde"

     //Dormir duerme por Numero segundos
     Dormir ::= "dormir" Numero

     ValorAbsoluto ::= "valor_absoluto" Numero
```

```
//Instruccion son las acciones que puede hacer Funcion, explicada mas adelante, difiere de las reglas de Programa en que esta tiene Devuelve, cosa que no necesita Programa

Instruccion ::= (Comentario | Declaracion | Expresion | Operandos | Devuelve | AccesoDatosComplejos)

//Forma de crear funciones, es requerido que toda función devuelva un Valor
Funcion ::= "funcion" Identificador Parametros? "inicio_funcion" (Instruccion+)? Devuelve "final_funcion."
     //En caso de múltiples parámetros, estos se dividen con espacios
     Parametros ::= "recibe"  (Parametro (" ")?)+
     Parametro ::= Tipo Identificador
     Devuelve ::= "devuelve" (Valor)? "\n"


//Manejo de Mapas y listas, el identificador ya debió ser declarado anteriormente y debe ser de tipo mapa o lista según corresponda
AccesoDatosComplejos ::= (DatosMapa | DatosLista) "\n"
     DatosMapa ::= (LlamarMapa | AsignarMapa | MapaLlaves | MapaValores)
          //Crea una llave
          LlamarMapa ::= Identificador "en" Valor
          //Asigna valor a una llave
          AsignarMapa ::= LlamarMapa "tiene" Valor
          //Cuenta cantidad de llaves, incluso las no asignadas
          MapaLlaves ::= Identificador "numero_llaves"
          //Cuenta cantidad de valores, no cuenta las llaves que aún no tienen un valor asignado
```

```
    MapaValores ::= LlamarMapa "numero_valores"
DatosLista ::= (AccesoAMemoria | LargoDeLista | MeterLista | SacarLista)
    //Obtiene un valor en una posición, el acceso inicia en la posición 0
    AccesoAMemoria ::= Identificador "acceder_en" Valor
    //Obtiene número de datos dentro de lista
    LargoDeLista ::= "obtener_largo" Identificador
    //Inserta un valor en la posición siguiente a la última posición actual de la lista
    MeterLista ::= Identificador "meter" Valor
    //Obtiene el Valor que se encuentre en la posición Numero de la lista, recordar que 0 es la primer posición
    SacarLista ::= Identificador "sacar_en" Numero
```

Ejercicios

Ejemplo 1: Sort

```
funcion bubblesort recibe lista elements inicio_funcion
    bool swapped tiene falso
    numerico largolista tiene obtener_largo elements
    numerico n tiene 0
    numerico i tiene 0
    desde n hasta largolista inicia_desde
        desde i hasta n inicia_desde
            si elements acceder_en i mayor elements acceder_en i mas 1 inicio_si
                swapped tiene verdadero
                elements acceder_en i tiene elements acceder_en i mas 1
                elements acceder_en i mas 1 tiene elements acceder_en i
            final_si
            i tiene i mas 1
        final_desde
        n tiene n mas 1
        si swapped igual falso inicio_si
            devuelve elements
        final_si
    final_desde
final_funcion
```

Ejemplo 2: Mapa

```
funcion recibirNombre inicio_funcion
  texto nombre tiene ""
  recibir_entrada con_comentario "digite el nombre" guardar_en nombre
  devuelve nombre
final_funcion

funcion recibirEdad inicio_funcion
  numerico edad tiene 0
  recibir_entrada con_comentario "digite la edad" guardar_en edad
  devuelve edad
final_funcion

numerico i tiene 0
mapa grupo de texto a numerico

mientras i menor_igual 10 inicia_mientras
  texto nombre tiene llamar recibirNombre
  numerico edad tiene llamar recibirEdad
  grupo en nombre tiene edad
  i tiene i mas 1
final_mientras

i tiene 0
numerico j tiene 0
desde 0 hasta grupo numero_llaves inicia_desde
  texto llave tiene grupo llaveen i
  desde 0 hasta grupo numero_valores inicia_desde
    escribir "El estudiante "
    escribir llave
    escribir " tiene la edad de "
    escribir grupo en llave
    escribir "años!"
    j tiene j mas 1
  final_desde
  i tiene i mas 1
final_desde

```

Ejemplo 3: Número primo

```
funcion primo recibe numerico num inicio_funcion
  devuelve llamar primoAux recibe num 2
final_funcion

funcion primoAux recibe numerico num numerico n inicio_funcion
  numerico numUsuario tiene 0
  recibir_entrada con_comentario "digite el numero" guardar_en numUsuario
  si numUsuario mayor_igual num inicio_si
    escribir "es primo"
    devuelve verdadero
  final_si
  sino si num residuo n diferente 0 inicio_si
        n tiene n mas 1
        devuelve llamar primoAux recibe num n
    final_si
    sino
        escribir "no es primo"
        devuelve falso
    final_sino
  final_sino
final_funcion

```

Ejemplo 4: Login Sencillo

```
// Este un ejemplo donde creamos un usuario y hacemos un login sencillo
mapa Usuarios de numerico a texto

// Funcion para crear usuarios en un mapa
funcion CrearUsuario recibe mapa Usuarios texto ComentarioID texto ComentarioNombre
inicio_funcion

  numerico ID tiene 0
  texto Nombre tiene ""

  recibir_entrada con_comentario ComentarioID guardar_en ID
  recibir_entrada con_comentario ComentarioNombre guardar_en Nombre

  Usuarios en ID tiene Nombre

final_funcion

// Aqui creamos los usuarios.
CrearUsuario Usuarios "Escriba el ID del primer usuario" "Escriba la contraseña del primer usuario"
CrearUsuario Usuarios "Escriba el ID del segundo usuario" "Escriba la contraseña del segundo usuario"
CrearUsuario Usuarios "Escriba el ID del tercer usuario" "Escriba la contraseña del tercer usuario"

// Vamos a hacer la funcion de login
funcion PuedeIniciar recibe mapa Usuarios texto ID texto Contraseña inicio_funcion

  texto Contra tiene Usuarios en ID
  bool inicia tiene falso

  si Contraseña igual Contra inicio_si
    inicia tiene verdadero
  final_si

  devuelve inicia
final_funcion


// Recibimos los datos del usuario y lo comparamos con los datos del mapa
numerico IDIniciar tiene 0
texto ContrasenaIniciar tiene ""

recibir_entrada con_comentario "Ingrese su ID: " guardar_en ID
recibir_entrada con_comentario "Ingrese su Contra: " guardar_en ContrasenaIniciar

bool inicia tiene llamar PuedeIniciar recibe Usuarios IDIniciar ContrasenaIniciar

// Dependiendo el resultado mostramos un comentario u otro.
si inicia igual verdadero inicio_si
  escribir "Bienvenido Ma fella :)"
final_si.

si inicia igual falso inicio_si
  escribir "Usuario o contraseña incorrectos"
final_si
```

Ejemplo 5: Fibonacci

```
1  // Fibonacci
2  funcion fibo recibe numerico n inicio_funcion
3    texto Contra tiene Usuarios en ID
4    bool inicia tiene falso
5
6    si n menor 2 inicio_si
7      devuelve n
8    final_si
9
10   numerico n1 tiene llamar fibo recibe n menos 1
11   numerico n2 tiene llamar fibo recibe n menos 2
12
13   devuelve n1 mas n2
14 final_funcion
```

Ejemplo 6: Operaciones aritméticas

```
// Funciones con las operaciones aritméticas
funcion suma recibe numerico x numerico z inicio_funcion
  devuelve x mas z
final_funcion

funcion resta recibe numerico x numerico z inicio_funcion
  devuelve x menos z
final_funcion

funcion multiplicacion recibe numerico x numerico z inicio_funcion
  devuelve x por z
final_funcion

funcion division recibe numerico x numerico z inicio_funcion
  si z igual 0 inicio_si
    devuelve "infinito"
  final_si
  devuelve x entre z
final_funcion

funcion modulo recibe numerico x numerico z inicio_funcion
  si z igual 0 inicio_si
    devuelve "infinito"
  final_si
  devuelve x residuo z
final_funcion

funcion potencia recibe numerico x numerico z inicio_funcion
  devuelve x elevado z
final_funcion
```

Ejemplo 7: Comparaciones

```
// Funciones haciendo uso de comparaciones
funcion mayorDeDosNumeros recibe numerico n1 numerico n2 inicio_funcion
  si n1 mayor n2 inicio_si
    devuelve n1
  final_si
  sino
    devuelve n2
  final_sino
final_funcion


funcion menorDeDosNumeros recibe numerico n1 numerico n2 inicio_funcion
  si n1 menor n2 inicio_si
    devuelve n1
  final_si
  sino
    devuelve n2
  final_sino
final_funcion

funcion sonNumerosIguales recibe numerico n1 numerico n2 inicio_funcion
  si n1 igual n2 inicio_si
    devuelve verdadero
  final_si
  sino
    devuelve falso
  final_sino
final_funcion

funcion sonNumerosDiferentes recibe numerico n1 numerico n2 inicio_funcion
  si n1 diferente n2 inicio_si
    devuelve verdadero
  final_si
  sino
    devuelve falso
  final_sino
final_funcion
```

Ejemplo 8: Con While

```
// Vamos a hacer la funcion de confirmar contraseña.
funcion confirmaContrasena recibe texto contrasenaInicial inicio_funcion

  numerico intentos tiene 3

  mientras intentos mayor 0 inicia_mientras
    recibir_entrada con_comentario "Confirme su contrasena: " guardar_en intentoContrasena

    si intentoContrasena diferente contrasenaInicial inicio_si
      intentos tiene intentos menos 1
      escribir "Contrasena incorrecta, quedan" intentos "intentos"
    final_si.
    sino
      escribir "Contrasena correcta, muchas gracias"
      devuelve true
    final_sino

  final_mientras

  escribir "Lo sentimos, no se pudo confirmar la contrasena"
  devuelve false
final_funcion.
```

Ejemplo 9: Manejo de texto

```
// Recibimos los datos del usuario y lo comparamos con los datos del mapa
texto nombre tiene "Bienvenido/a a McDonald's "
texto textoTemp tiene ""

recibir_entrada con_comentario "Ingrese su nombre: " guardar_en textoTemp
nombre tiene nombre mas textoTemp
nombre tiene nombre mas " "
recibir_entrada con_comentario "Ingrese su apellido: " guardar_en textoTemp
nombre tiene nombre mas textoTemp
nombre tiene nombre mas " "
recibir_entrada con_comentario "Ingrese su otro apellido: " guardar_en textoTemp
nombre tiene nombre mas textoTemp
nombre tiene nombre mas ". ¿Qué le puedo ofrecer hoy?"
escribir nombre
// Pedir la orden de mcDonald's
//. . .
escribir "Las opciones de pago son: "
escribir "-> Visa " mas " Platinum " por 2
escribir "-> Visa " mas " Platinum " por 3 mas " Plus"
escribir "-> Visa " mas " Platinum " por 3 mas " Plus" por 10
```

Ejemplo 10: Función Fizzbuzz

```
1  // Funcion Fizz Buzz
2  funcion fizzBuzz recibe numerico n inicio_funcion
3    si n modulo 3 igual 0 inicio_si
4      si n modulo 5 igual 0 inicio_si
5        devuelve "FizzBuzz"
6      final_si
7      sino
8        devuelve "Fizz"
9      final_sino
10   final_si
11   sino
12     si n modulo 5 igual 0 inicio_si
13       devuelve "Buzz"
14     final_si
15     sino
16       devuelve n
17     final_sino
18   final_sino
19 final_funcion
20
```

Ejemplo 11: Función Generador de números Pseudo Aleatorios

```
funcion pseudo_aleatorio recibe numerico semilla inicio_funcion
  numerico a tiene semilla por 15485863
  devuelve a por a por a modulo 2038074743
final_funcion

numerico x tiene pseudo_aleatorio recibe numero_aleatorio de 0 a 1
numerico z tiene pseudo_aleatorio recibe numero_aleatorio de 0 a 1

```

Ejemplo 12: Número más alejado del cero

```
funcion masAlejadoDeCero recibe numerico a numerico b inicio_funcion
  aAbs tiene valor_absoluto a
  bAbs tiene valor_absoluto b
  si aAbs igual bAbs
    inicio_si
      escribir "Igualmente alejados de cero"
      devuelve a b
    final_si
  sino si aAbs mayor bAbs
    escribir "El primer numero está más alejado"
    devuelve a
  sino
    escribir "El segundo numero está más alejado"
    devuelve b
final_funcion
```

Completitud

Una subevaluación del proyecto se puede representar en esta sección, donde se colocan todas las funciones que debe o puede incluir la entrega de este proyecto y se marcan un “check” si este los tiene presentes.

- ☑ ~~Motivación~~
- ☑ ~~Puntos fuertes~~
- ☑ ~~Gramática~~
- ☑ ~~Ejemplos de código en su gramática~~
- ☑ ~~Lenguaje en español~~
- ☑ ~~Estructura de repetición~~
- ☑ ~~Estructura de bifurcación~~
- ☑ ~~Soporte para variables locales y globales~~
- ☑ ~~Soporte para funciones~~
- ☑ ~~Soporte para comentarios~~
- ☑ ~~Soporte para textos, números enteros y flotantes~~
- ☑ ~~Expresiones matemáticas~~
- ☑ ~~Expresiones condicionales~~
- ☑ ~~Funciones de ambiente estándar~~
- ☑ ~~Colores personalizados (extra)~~
- ☑ ~~Ejercicios extras (extra)~~

Conclusiones

Se llegó a la comprensión grupal de que se proyecta llegar hacia un lenguaje de programación que facilite la codificación mediante la escritura de comandos que un lector de pantalla pueda leer y que permita entender auditivamente el flujo de ejecución de un programa.

Para esto se aprendió que el uso de signos especiales sí es necesario, a pesar de que en un inicio se pretendió no usarlos, esto pues así se entiende de mejor manera el código al hacer que el lector de pantalla haga pausas.

Se puso en práctica la redacción de ejemplos de código haciendo uso de nuestra propia gramática EBNF; en el proceso, nos dimos cuenta de la necesidad tan importante de llevar la teoría a la aplicación para ser capaces de notar aspectos de mejora en nuestra gramática y trabajar en las debidas actualizaciones. En dicho proceso, también entendimos que un pequeño cambio en la gramática, puede desencadenar conflictos lógicos y/o sintácticos, y a su vez, entorpecer el progreso que se había obtenido hasta ese momento con los ejemplos de código ya creados.

Referencias

Instituto ORL-IOM. (2016, junio 6). *Los 7 animales con el mejor oído del planeta*. Instituto ORL-IOM; Instituto ORL IOM - Grupo Antolí Candela. https://www.institutoorl-iom.com/blog/los-7-animales-con-el-mejor-oido-del-planeta/

Gimenez. (2017, julio 19). *¿SABÍAS QUE... LOS BÚHOS SON HIPERMÉTROPES?* Doctor Giménez.Instituto Oftalmológico. https://www.doctorgimenez.com/noticias/sabias-los-buhos-hipermetropes/

Porter, D. (2022, agosto 4). *¿Qué es la hipermetropía?* American Academy of Ophthalmology. https://www.aao.org/salud-ocular/enfermedades/hipermetropia